Using eBPF as a heterogeneous processing ABI

Jakub Kicinski <kuba@kernel.org>
Motivation

Open up the HW

"switchdev" XDP

control processing pipelines
BPF Sandbox

- As a goal of BPF IR JITing of BPF IR to most RISC cores should be very easy
- BPF VM provides a simple and well understood execution environment
- Avoids leaking implementation details into the definition of the VM and ABIs (the abstraction benefits kernel as much as accelerators)
- Unlike higher level languages BPF is a intermediate representation (IR) which provides binary compatibility, it is a mechanism
- BPF is extensible through helpers and maps allowing us to make use of special HW features (when gain justifies the effort)

Make it easier for vendors to add BPF offload for I/O devices which increasingly take a form of slightly customized RISC cores.
Compare to other IRs

- High level language: OpenCL C, GLSL, BPF C, P4, VHDL
- Intermediate representation: SPIR-V, TGSI, NIR, eBPF, P4CIR
- Primitives: LD/ST + math ops, RISC ops, parse/table/action
- Targets: GPUs, RISC, switch ASIC, logic

- all models support some call out/black box invocation
- all models fit into a fixed pipeline
- other models declare variable types (not that it matters..)
- SPIR-V supports multiple memory models
- graphics stacks usually allow use of complex math instructions
- other IRs try to be lossless/preserve semantics for longer
- other JITs require a full compiler to go from IR -> code
## Option 1 - JIT reuse

Have the CPU compile machine code to load.

**Core work:**
- Untangle JITs from architectures
- Ensure PIC (or record relocations)

**Driver must have:**
- List of supported context fields
- Helper addresses
- Map ID/ptr to use

**Potentially needed:**
- Size/offset of context fields

**Hopefully not needed:**
- Different calling convention
- Different register mapping

## Option 2 - IR handoff

Send the BPF IR down to the device.

**Advantages:**
- No trust required
- Simpler driver

**Disadvantages:**
- HW devices (not paravirt) rarely run full Linux
- Code duplication
- Closed source FW

IR handoff can be implemented at higher layer by user space requesting the load via hypervisor service.
Potential reuse of JITs

- BPF
- x86
- Arm
- MIPS

Verifier checks:
- install map pointers
- check CFG
- check paths
- remove dead code
- check max stack
- gen prologue
- rewrite CTX access
- rewrite DIV/MOD insns
- rewrite legacy access
- inline calls
- resolve calls
- blind constants
- prologue
- code gen
- epilogue

Verifier rewrite:
- ✓
- ✓
- ✓
- ✓
- ✓
- ✓
- ✓
- ❌
- ○
- ○
- ○
- ○
- ○

Current NFP offload:
- ✓
- ✓
- ✓
- ✓
- ✓
- ✓
- ✓
- ✓
Quick PoC

x86_64
BPF core

arm64
bpf_jit_comp.c

netdevsim

defaults

xdp_buff + helpers

user space

linker

QEMU
Aarch64

# hexdump -v -e '/1 "0x%02x, "' \ $netdevsim0_dfs/arm_asm > raw_asm

# make
aarch64-elf-gcc -g -c env64.c -o env64.o
aarch64-elf-as -g -c startup64.s -o startup64.o
aarch64-elf-ld -Tenv64.ld env64.o startup64.o -o env64.elf

# qemu-system-aarch64 -M virt -cpu cortex-a57 \
   -nographic -kernel env64.elf
Env starting up...
  Entering XDP prog (pkt len: 4096)
    adjust head: 12
  After XDP prog (ret: 2, pkt len: 4084)
Thank you!

Discussion

*(how) do you think eBPF can help open the hardware?*